Sensorium – An Active Monitoring System for Neighborhood Relations in Wireless Sensor Networks

Stefan Nürnberger, Reinhardt Karnapke, and Jörg Nolte

Brandenburg University of Technology Cottbus
{snuernbe,karnapke,jon}@informatik.tu-cottbus.de

Abstract. Communication neighborhood in wireless sensor networks changes often as links break or appear. Therefore, monitoring link quality and (logical) network topology is necessary. As node placement has a large influence on the radio neighborhood and its changes, different positions should be evaluated before starting the actual application.

In this paper we introduce Sensorium, an active monitoring system that supplies the user with an insight into the neighborhood relations between nodes and their changes in time. It can be used before the actual deployment to evaluate different possibilities of node placement and choose the one that offers the best connectivity.

Keywords: wireless sensor networks, active monitoring, robustness, Sensorium.

1 Introduction

Wireless sensor networks offer a wide variety of applications which merit a number of design goals. When a sensor network is designed and deployed, network lifetime is one of the major goals. But there are two factors that need to be considered during deployment: Sensor coverage and network connectivity. The application dictates the area that has to be surveyed, thus deciding the minimum number of nodes needed to reach the desired coverage. To determine the needed number of nodes to reach the necessary connectivity, it has to be measured in a pre-deployment phase. But measuring it only once is not enough. As various experiments have shown, network connectivity varies a lot over time [20,18,9,6,8,5]. These changes make it necessary to monitor the connectivity either constantly while the application is running, or for a sufficient time before the deployment. Monitoring during network operation always introduces runtime overhead and active monitoring changes the surveyed system. For this reason, we decided to use a pre-deployment monitoring approach.

In this paper we present Sensorium, a monitoring system for neighborhood relations in wireless sensor networks. It collects the information of each node placed in a location that could be used in the real deployment before the sensing application is started. The connectivity information is then gathered at a central

J. Zheng, D. Simplot-Ryl, V.C.M. Leung (Eds.): ADHOCNETS 2010, LNICST 49, pp. 34–47, 2010.

sink, enriched with a map of the deployment area and presented visually. Based on this information, a user can decide where to add additional nodes or to move already deployed ones, in order to have the best connectivity possible.

This paper is structured as follows: Section 2 shows different approaches that can be used when designing a monitoring system for sensor networks. Our system, SENSORIUM, is presented in section 3 and an exemplary deployment described in section 4. Related work is given in section 5. We finish with a conclusion in section 6.

2 Designing a Monitoring System for Wireless Sensor Networks

When designing a monitoring system for wireless sensor networks, it is always necessary to have at least a partial knowledge of the intended application. Designing a general monitoring system that can be used for all applications is nearly impossible and will automatically lead to a waste of resources when data is gathered that is not needed. Therefore it is mandatory to define the monitoring issues first. These can include for example the liveliness of the sensor network as a whole or individual node failure. Connectivity, i.e., the ability of the sensors to deliver the sensed values to the sink is another critical aspect, even though there is not much that can be done to change it after deployment. Sometimes sensors on individual nodes will fail, delivering false data, which can also be detected on the sink, , e.g., by comparing the values detected by neighboring nodes. If the difference is too high, one or more of the sensors must be faulty. The energy status of nodes can be measured and used for traffic shaping to prevent node failures. After node failures, the human operator will need to know if the area that has to be monitored is still completely covered.

Some of these monitoring goals can be reached without transmitting additional data, simply by deriving information from the traffic that arrives at the sink. Others require the monitoring system to generate its own information on the nodes and transmit it to the sink in one way or another. This second approach is called an intrusive way or an active monitoring system while systems that use the first approach are called passive monitoring systems.

2.1 Passive Monitoring

Passive monitoring systems aim to provide all information desired by the user by deducting it from the messages that are transmitted by the application anyway. Sources of information are , e.g., the number of packets that arrived at the sink either from a certain node or from a certain part of the network. When the routing topology (, e.g., a tree) is known, it is possible to find node failures that are responsible for missing data from a whole subtree. Another possible source of information is the application data collected. If the values collected from one particular node are not within a certain boundary or diverge too much from those of neighboring nodes, the probability of sensor failure is high.

The big advantage of passive monitoring is that it does not interfere with the application in any way. Timing, network load and energy consumption remain the same whether it is used or not. It is also possible to make a post-mortem analysis of the network from stored data. There are, however, a number of disadvantages. When no messages from a node arrive at the sink, it is not possible to distinguish between the different possible reasons: Packet loss, congestion, link failure and node failure all lead to the same error pattern. Another disadvantage is that only a limited view of the status of each node is provided. If the application data does not include the energy status of the nodes, it can only be roughly approximated. Neighborhood information and area coverage are also problematic. The most important problem, though, is that passive monitoring is only usable for applications that communicate regularly, like , e.g., habitat monitoring. When the application does not communicate regularly, it is not possible to satisfy the liveliness criterion needed , e.g., by networks deployed for fire protection or intrusion detection.

2.2 Active Monitoring

There are two ways of realizing active monitoring, in band and out of band. Both of them can be used to gather periodic data or request data on demand. The big advantage of active monitoring is that any kind of data that could be of interest can be monitored. The biggest disadvantage is that it interferes with the normal operation of the sensor network. If it is realized with in-band communication (i.e., using the same means of communication as the application) it increases the network load which can lead to congestion and thus packet loss. Even if no packet loss occurs, it can influence the timing behavior badly. If additional communication hardware is used, the energy consumption of the sensor nodes is still increased. Even if the additional hardware comes with its own power supply, it still needs to access at least the memory to retrieve , e.g., neighborhood tables. Also, the additional hardware makes the nodes more expensive. All this influence on the node behavior can either mask errors or even introduce new ones.

Even though there are so many disadvantages that need to be considered, active monitoring is absolutely necessary for a number of criteria. When liveliness is the number one goal, and failing nodes have to be replaced immediately, active monitoring should be used. It is also quite useful for monitoring of the logical topology of a network. This enables the evaluation of possible locations for node placement in a pre-deployment phase, to find the minimum number of nodes needed and their placements to reach the desired area coverage.

3 The Architecture of Sensorium

Sensorium was developed as a tool that enables the user of a wireless sensor network to find out the best placement and the number of nodes required to ensure connectivity between nodes and the sink. To supply this information, Sensorium uses the active monitoring approach, and gathers extensive information

about the status of every node. While this would interfere with the normal operation of a sensor network, SENSORIUM is meant to be used in a pre-deployment phase, where no application is running yet. It could also be used after deployment, but then all the disadvantages of active monitoring described above would take effect.

In order to provide information about network connectivity, a simple neighborhood discovery protocol is used: all nodes transmit messages regularly. Upon reception of such a message a node notes that at the current point in time there had been an incoming link. To retrieve this information, the sink node sends info request messages into the network, which are answered by the nodes with their current state information. The network behavior is therefore similar to that of a typical sense-and-send application, where a powerful sink requests data from the sensor nodes, which is then merged and stored for further evaluation.

In SENSORIUM, no data packet aggregation is used on the forwarding nodes, because SENSORIUM already aggregates neighborlists, routing metrics, sensor data and all other requested information into a single packet. Such a packet is already fairly large. As these types of data can not be combined without losses and aggregating two messages would double the size, the probability of packet loss would increase too much.

SENSORIUM uses a modular architecture which allows a user to easily exchange routing protocols or the evaluation interface and add new information sources (, e.g., new types of sensors). This modular architecture makes it also easy to switch between hardware platforms, and even allows the usage of one programming language on the sensor nodes and another on the sink or the host system where the results are displayed for the user.

3.1 Notion of Chronological Sequence

When monitoring network connectivity, it is necessary to be able to identify links between nodes, and their changes in time. Therefore some notion of time is needed, to build a row of snapshots of the logical network topology. In SENSORIUM time is divided into epochs, with each node collecting its neighborhood information anew in each new epoch. Clock skew is tolerated. This is possible because the nodes always send their current epoch value with their answers, enabling the sink to calculate the skew from its own epoch counter. The skew is used when requesting values from the past that have been stored on the sensor nodes (see data preparation at the sink, section 3.4). Please note that choosing the right length of the epoch is crucial for the success of SENSORIUM. If it is too long, changes will be missed. If it is too short, the network load gets too high, possibly resulting in congestion. Also, determining the connectivity takes time, as the neighborhood discovery protocol needs to transmit its messages. Epochs must be larger than this amount of time.

3.2 Message Routing

The need to transmit data back to the sink in an ever changing radio neighborhood makes the usage of an adaptive routing protocol mandatory. The choice of

routing protocol is heavily influenced by the communication scheme exhibited by SENSORIUM: One to many for broadcasts from the sink to all nodes (requests), many to one for replies from the sensor nodes to the sink, and one to one for history requests from the sink to a node and the answers. For the first case, transmission from the sink to all sensor nodes, a simple flooding (with duplicate suppression) is used. This ensures the highest delivery ratio possible (if any path to a node exists, it will receive the message) and does not introduce much overhead, as all nodes need to receive the message anyway. The overhead measured in data packets could be reduced a little by having only a subset of nodes retransmit the messages, , e.g., using multi point relay nodes like OLSR [4] does. But finding these nodes and keeping the information up to date would introduce unwanted protocol overhead, which could get quite large if the rate of changes increases. The second case, where the sensor nodes transmit to the sink, has to be handled like a one to one communication though, because no aggregation can take place on the intermediate nodes for reasons described above. Please note that the second case is now identical to the third one, and we will use the same protocol for both cases. For the third case we need a reactive routing protocol that does not rely on topology information, as we are only collecting that information now. Occasional packet losses can be tolerated, as the history function of SENSORIUM is used to automatically re-request missing data. Therefore, a best effort protocol with a fairly high probability of successful delivery is sufficient.

In previous work we introduced Buckshot Routing [13], a robust source routing protocol for wireless sensor networks. In Buckshot Routing, a node that receives a message does not only forward it if its own identity is enclosed, but also if one of its neighbors is on the path the message has yet to travel. This leads to a broader tunnel (with a breadth of one hop) around the source route along which the message travels, circumnavigating broken or unidirectional links, failed nodes and other obstacles. Sometimes the links that are used to circumvent these problems are unidirectional themselves, but pointing the other way. Buckshot routing has a delivery ratio that is close to that of flooding while keeping the number of involved nodes much smaller, especially in large networks. We use Buckshot Routing for all communication from sensor nodes to the sink in SENSORIUM. The original path around which Buckshot Routing builds its tunnel is obtained when the request from the sink is flooded into the network. Each node that forwards the message attaches its own identity, collecting the path that has been taken. The inverted path is then used for the answers from each sensor node to the sink.

Duplicate detection is realized using the identity of the sender and a sequence number defined by the sink. If a node is reset for any reason, it still receives the current sequence number with the next request from the sink, avoiding all false positives in duplicate detection.

3.3 The Request Handler

The request handler which runs on all sensor nodes is used to evaluate the queries sent by the sink. It works on a configurable set of information sources which are represented as bits in a bit field in the query messages. This representation

enables the sink to specify different sources within each query. The answers are represented in type-length-value (tlv) encoding, which eases the inclusion of new sources by simply adding a new type and specifying a position in the bit field.

In our examples the sources used are neighborhood details, routing information, packet statistics, temperature, light intensity and current energy level. The request handler is realized as a service that decodes the requests, compiles the requested information from its sources and transmits the response messages. The sink then combines the information from all nodes to a snapshot of the status of the whole network.

3.4 Data Preparation at the Sink

The sink runs a local version of almost all node components. The routing is a little different, though. Instead of forwarding packets to another node, they are collected and evaluated here (replies) or injected into the sensor network (requests). The sink provides some of the information sources just like the other sensor nodes (neighborhood information, packet statistics) but omits some others (no sensors, no battery). It also does not have to store a local history as requests to the sink should never fail.

There are two modes of operation for the sink, idle and monitoring. In idle state the sink simply waits for the first requests from the user to switch into the active, monitoring mode. Once it has been switched to monitoring, it sends out the periodic information requests to all nodes and keeps the information up to date. This is done by collecting all incoming data, sorting it according to the epochs and identifying nodes from which data is still missing.

The counter for missed responses from those nodes is increased, if it reaches a certain threshold the nodes are marked as currently unreachable. At the same time, a history message is created and entered into the history queue for those nodes not marked as unreachable. The messages contained therein are sent repeatedly at regular intervals, until an answer is received or the node in question is marked as unreachable. Once any message is received from a node that has been marked as unreachable before, it is marked as reachable again and a history request for all missed data is generated and entered into the queue. To enable this recovery of lost data, the sensor nodes need to store it locally for a certain amount of time. This is realized using a ring buffer in which the last n values are stored. Which sources are included in the ring buffer is configurable, it does not have to be all that are monitored.

The sink also offers an application programming interface which can be used to control the monitoring process (, e.g., define which sources should be requested in the queries) and request the collected information of a whole epoch (involves no communication inside the sensor network). It is the general interface for user applications.

3.5 The User Interface

The user interface is used to switch the modes on the sink between idle and monitoring, and to define the desired information sources for the queries. It

also offers the possibilities to configure the presentation and evaluation of the gathered data, as all data is transmitted from the sink to the user interface unevaluated. Figure 1 shows an example of the web interface which is used to gain fast access to listings of all data already gathered and the visualization of the logical topology of the network (radio neighborhood). The web interface can run on almost every platform, even on mobile devices like smart phones for on-site evaluation as it only requires a web browser.

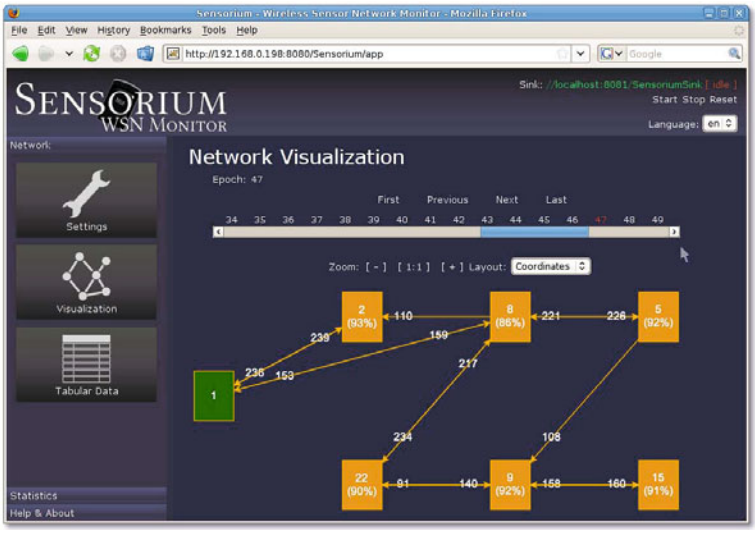

Fig. 1. The Web Based User Interface of Sensorium

Seeing the logical topology is already helping the user a great deal, but the actual geographic information is still necessary to decide which nodes may be removed while keeping the area coverage required by the application. Therefore, a GIS based interface is also provided. It visualizes the network as graphic overlay, and can use data from any geographic database. This is especially useful for deployments like , e.g., fire detection in a nature-sanctuary, where a map taken by airplane can be combined with the GPS coordinates delivered by the nodes. The GIS interface can also visualize the sensor values (, e.g., temperature or light intensity) and the neighborhood- and routing information. All gathered data that is not represented directly is easily accessible from here. An example of the visualization taken from our outdoor experiments is shown in figure 3, section 4.3.

4 Deploying Sensorium

To evaluate our prototype implementation, we made a number of experiments which are described in this section, after the hard- and software we used is presented.

4.1 Hardware

We employed the SunSPOT nodes developed by Sun Microsystems as the sensor node platform for our implementation of SENSORIUM. The SunSPOT hardware consists of a baseboard with the computing and communication parts and a stacked sensor board with the sensor hardware and additional connectors. The baseboard features a 32bit ARM920T processor, 512kB of RAM and 4MB Flash memory as well as a 2.4GHz IEEE 802.15.4 compatible radio module. The sensor board includes temperature and light sensors as well as a three axis accelerometer, eight LEDs and two user buttons. External components can be connected to the sensor board through six analog inputs and five general purpose I/O pins. For the determination of a node's physical position, we connected a SiRF-III based GPS module to each node.

Fig. 2. One SunSPOT with GPS Module and a Base Station

A consumer grade laptop is used as the sink node in our deployments. It is equipped with a 1GHz PowerPC G4 processor, 768MB of RAM and has an attached SunSPOT base station for communication with the wireless sensor nodes. The base station consists only of the SunSPOT base board. No sensor board or battery is attached to it. A SunSPOT node with GPS module and the base station are shown in figure 2.

4.2 Software

The whole software for SENSORIUM is written in the Java programming language because of the used SunSPOTs platform. If any other hardware was used, it would be no problem to implement the necessary components for the sensor nodes in , e.g., C++ as only the packet types have to be adhered to. We have chosen the SunSPOTs because we wanted to evaluate neighborhood relations on sensor nodes equipped with an IEEE 802.11.4 transceiver. This was our first evaluation of the SunSPOT software development kit from Sun Microsystems.

The SunSPOTs run a lightweight Java Virtual Machine, which enables rapid application development for wireless sensor networks in a well known programming language. The software for the sensor nodes and the sink depends on the SunSPOT SDK. Communication between sink and user interfaces (not within the sensor network) is based on Java RMI and is thus uncoupled from the SDK. While this design enables the use of dedicated machines for data gathering and evaluation, in our field experiments those services were consolidated on one machine. The system also allows for concurrent access to the gathered data from different user interfaces. The central data management on the sink ensures that the view on the network is consistent for all users even across different kinds of user interfaces.

The web interface for SENSORIUM is a Rich Internet Application based on the open source Echo Web Framework [12] which allows interactive web applications to be written completely in Java. In our installation the application is hosted on an Apache Tomcat Web Server [10]. Concurrent access to this interface is possible, while the application manages independent sessions for each user. The clients need only be equipped with a reasonably modern web browser with JavaScript support. A Java installation or Java RMI are not needed.

The implemented GIS interface is based on uDig [16], a standards compliant open source geographic information system. Since uDig itself is based on the Eclipse Rich Client Platform [17], the application is platform independent and easily extensible with custom plugins.

4.3 Experiments

In the first row of field experiments we deployed networks in varying size from 6 to 25 nodes. The requested data included detailed neighborhood information, energy level and the geographical position of the nodes determined from the attached GPS modules. Unfortunately we experienced an unusually low communication range. The sensor nodes were only able to communicate reliably over the fair distance of five to seven meters. Since this is also approximately the accuracy of the GPS modules, the received geographic coordinates were quite useless to determine the physical layout of the sensor network. The difference to the experiences of others that were able to communicate over a distance of up to 70 meters [23] with the same hardware was probably caused by a high noise level in the 2.4GHz band at the deployment site. While the exact source of this noise is unknown, we were forced to reduce the covered area of our deployments.

The experiments have shown that it is almost impossible to predict the network topology of an ad hoc deployment. Our topology varied between a very dense network and complete network separation. We also experienced a very unbalanced neighbor count among the nodes despite the regular grid layout we employed in our field experiments. Through the use of SENSORIUM in the pre-deployment phase we were able to track the topology changes almost immediately and adjust the topology accordingly. A real world deployment without the use of a live monitoring system would have been fatal under the mentioned circumstances.

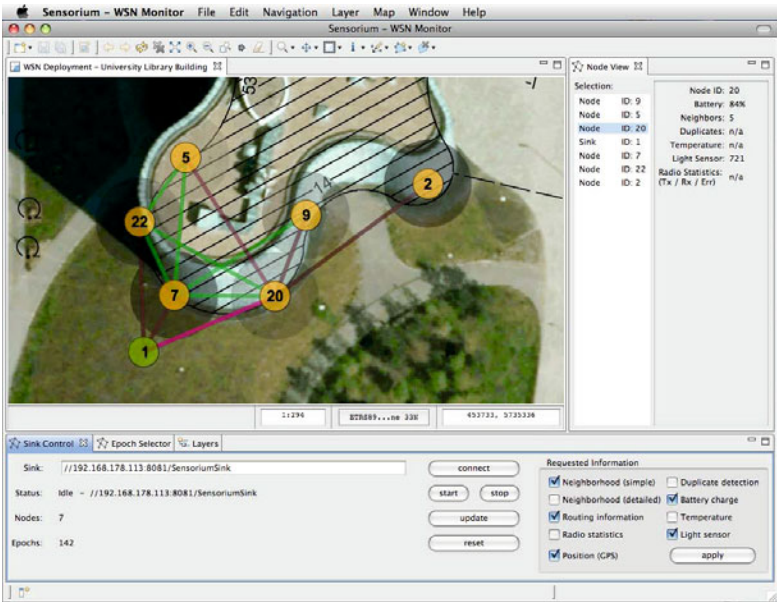

Fig. 3. Deployment at the University Library

The topology visualization through the web interface of SENSORIUM is good for small networks, but got quite cluttered with increasing node and neighbor count. More sophisticated graph layout algorithms could help diminish that problem. Later analysis of the log files on the sink showed that a considerable amount of information was retrieved from the nodes' history buffers. Without those the achieved exhaustive topology view would not have been possible.

In a later experiment with 6 nodes the GIS based user interface was evaluated. As different deployment site the university library building was chosen. The requested data included neighborhood information, energy level, the nodes' geographic position, their current routing information and the readings from the built in light sensors. Again, some of the information had to be retrieved from history buffers. A screenshot from this deployment is shown in figure 3. The gathered information from the sensor network fits nicely into an overlay of geographic data like the shown digital orthophoto of the area.

An automated statistical evaluation of the gathered data is still missing, but may be integrated into the existing user interfaces in the future. The history management mechanism has proven to be an essential part of the system that is able to compensate the insufficiencies of the unreliable wireless communication.

5 Related Work

Pimoto [1] is a completely passive monitoring system. It relies on dedicated monitoring nodes that are deployed within the monitored network. Those nodes

capture the traffic from the network and push it to a central server. Bluetooth communication is used to avoid transmitting the data in-band where it could influence the monitored system. Through this design the monitoring does not need to be suspended while data is transferred to the sink. Of course, this only works when the sensor network operates in another frequency band than Bluetooth. The captured traffic is enhanced with meta data such as the capture time and the monitoring node's address. The server combines the data from all nodes and automatically reorders packets with the information provided in the meta data. Display and analysis of the traffic generated by the sensor nodes is accomplished through a custom plugin for Wireshark, a popular network monitoring tool formerly known as Ethereal [7]. Pimoto may be described as a packet sniffer for wireless sensor networks that can be used for protocol debugging. The developers emphasize that due to the redundant hardware the system may be used to analyze already deployed networks. But the completely passive operation renders it unsuitable for applications that exhibit little or no traffic at all. It also means that in contrast to SENSORIUM Pimoto is not able to collect information about radio neighborhood.

The authors of Sympathy [14,15] utilize another approach to monitoring. They use a concept of metrics to identify whether the monitored network is in a state of error. Their primary metric is the number of packets that arrive at the sink. Applications that yield the expected amount of data are likely to be working correctly. This way the impact on bandwidth and energy consumption is minimized. For applications that are not expected to communicate, Sympathy offers the possibility to let nodes actively transmit periodic metrics like neighbor count and current parent in the routing topology to the sink. When the metrics suggest a state of error, e.g., some node repeatedly fails to transmit data, Sympathy systematically inquires additional metrics from the nodes on the failed path. Since single node failures in the sensor network may lead to subsequent faults, when , e.g., the network gets separated, Sympathy employs an empirical decision tree to identify the root cause of each detected failure. This way the human operator of the network is presented with the most likely cause for the detected behavior and may induce direct action. Sympathy is primarily thought of as a sensor network debugger. As such it gives little insight to the workings of a functional network. Comprehensive information is only generated when a fault is assumed. Furthermore, a good deal of information about the expected application behavior and network topology has to be provided in order to yield accurate results. As Sympathy uses the messages generated by the application to get an insight into the state of the network, it can not be used in pre-deployment like SENSORIUM.

PAD [11] is a system similar to Sympathy that omits the active inquiry part and uses a packet marking algorithm instead. Each packet transmitted through the network is marked with a hop count and one of the forwarding nodes' id. The sink builds a belief network from this information, which is in turn used to identify failures and their causes. The authors claim an accuracy of about 90% for the fault identification algorithm. Since the correctness of the belief network

converges slowly with the number of received packets, the approach is less likely to work for frequently changing topologies.

For sensor networks based on TinyOS, a nesC component framework for active network monitoring and management called SNMS [19] can be used. It is intended to be deployed alongside a sensor network application. SNMS provides a flexible, application cooperative means to gather information about the state of the sensor nodes and execute basic commands like node reset. Application programmers may define additional parameters that are exported through the system. Interpretation of this data is left to the network operator. SNMS offers query based health monitoring of the deployed applications as well as an event logging system. The nodes log events locally and transmit them to the sink when a user defined threshold is exceeded. SNMS has also been extended with an RPC mechanism that enables the management system to execute arbitrary commands of the deployed application. The extended system is called L-SNMS [22] and offers an enhanced Java based user interface for the visualization of events in the network and additional management functionality such as reprogramming of nodes. L-SNMS is the only system of those described above that offers a visualization of the gathered information, which in our opinion is absolutely necessary to help the user understand what is happening inside the monitored sensor network. Therefore, SENSORIUM offers multiple user interfaces which show the logical and/or geographical topology of the network enriched with further gathered information.

Different tools for sensor network visualization such as WiseObserver [3], Mote-View [21] or SpyGlass [2] are available. While WiseObserver and Mote-View visualize data available from a connected database, they are not concerned with obtaining this data in the first place. SpyGlass uses a simple flooding based scheme to periodically broadcast sensor readings to gateway nodes. All three systems focus on the efficient visualization of routing information and sensor readings. SpyGlass also supports the creation of custom specialized visualization plugins. In contrast to these protocols, SENSORIUM offers a complete solution, gathering data in the sensor network, taking care of lost messages, enriching the gathered data with freely available terrain information and presenting the processed information to the user.

6 Conclusion

In this paper we have presented SENSORIUM, an active monitoring system for wireless sensor networks. It is meant to be used in a pre-deployment phase, to evaluate possible placements of individual sensor nodes. SENSORIUM actively collects neighborhood information from the sensor nodes to deliver a complete view of network connectivity to the user. When this is evaluated over a certain time, a good placement for the sensor network can be found. We have shown that our prototype implementation works in real world experiments with SunSPOT sensor nodes. An interesting fact discovered in one of them is that the communication range we measured in the first deployment was only a few meters

while other researchers using the same hardware were able to reach up to 70 meters. If the sensor network had been deployed without active monitoring, the insufficient connectivity would have been discovered much later, leading to loss of application data.

References

1. Awad, A., Nebel, R., German, R., Dressler, F.: On the need for passive monitoring in sensor networks. In: Proceedings of the 11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools, pp. 693–699. IEEE Computer Society, Los Alamitos (2008)
2. Buschmann, C., Pfisterer, D., Fischer, S., Fekete, S.P., Kröller, A.: Spyglass: a wireless sensor network visualizer. SIGBED Rev. 2(1), 1–6 (2005)
3. Castillo, J.A., Ortiz, A.M., López, V., Olivates, T., Orozco-Barbosa, L.: Wiseobserver: a real experience with wireless sensor networks. In: PM2HW2N 2008: Proceedings of the 3nd ACM Workshop on Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks, pp. 23–26. ACM, New York (2008)
4. Clausen, T., Jacquet, P.: Optimized link state routing protocol (olsr) rfc 3626 (2003)
5. Couto, D.S.J.D., Aguayo, D., Bicket, J., Morris, R.: A high-throughput path metric for multi-hop wireless routing. In: MobiCom 2003: Proceedings of the 9th Annual International Conference on Mobile Computing and Networking, pp. 134–146. ACM, New York (2003)
6. Dinh, T.L., Hu, W., Sikka, P., Corke, P., Overs, L., Brosnan, S.: Design and deployment of a remote robust sensor network: Experiences from an outdoor water quality monitoring network. In: LCN 2007: Proceedings of the 32nd IEEE Conference on Local Computer Networks (LCN 2007), Washington, DC, USA, pp. 799–806. IEEE Computer Society, Los Alamitos (2007)
7. Hards, B.: A guided tour of ethereal. Linux Journal (118), 7 (2004)
8. He, T., Krishnamurthy, S., Luo, L., Yan, T., Gu, L., Stoleru, R., Zhou, G., Cao, Q., Vicaire, P., Stankovic, J.A., Abdelzaher, T.F., Hui, J., Krogh, B.: Vigilnet: An integrated sensor network system for energy-efficient surveillance. ACM Trans. Sen. Netw. 2(1), 1–38 (2006)
9. Langendoen, K., Baggio, A., Visser, O.: Murphy loves potatoes: Experiences from a pilot sensor network deployment in precision agriculture. In: Proc. 14th Intl. Workshop on Parallel and Distributed Real-Time Systems (WPDRTS) (April 2006)
10. Lerner, R.M.: At the Forge: Server-Side Java with Jakarta-Tomcat. Linux Journal 2001(84), 10 (2001)
11. Liu, K., Li, M., Yang, X., Jiang, M.: Passive diagnosis for wireless sensor networks. In: SenSys 2008: Proceedings of the 6th ACM conference on Embedded network sensor systems, pp. 371–372. ACM, New York (2008)
12. NextApp Inc., Echo Web Framework, `http://echo.nextapp.com/site/echo3`
13. Peters, D., Karnapke, R., Nolte, J.: Buckshot routing - a robust source routing protocol for dense ad-hoc networks. In: Ad Hoc Networks Conference 2009, Niagara Falls, Canada (2009)
14. Ramanathan, N., Chang, K., Kapur, R., Girod, L., Kohler, E., Estrin, D.: Sympathy for the sensor network debugger. In: SenSys 2005: Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems, pp. 255–267. ACM, New York (2005)

15. Ramanathan, N., Kohler, E., Girod, L., Estrin, D.: Sympathy: A debugging system for sensor networks. In: LCN 2004: Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks, pp. 554–555. IEEE Computer Society, Los Alamitos (2004)
16. Ramsey, P.: User Friendly Desktop Internet GIS (uDig) for OpenGIS Spatial Data Infrastructures. Technical report, Refractions Research Inc. (2003), `http://udig.refractions.net/docs/udig-summary.pdf`
17. Rubel, D.: The heart of eclipse. Queue 4(8), 36–44 (2006)
18. Sang, L., Arora, A., Zhang, H.: On exploiting asymmetric wireless links via one-way estimation. In: MobiHoc 2007: Proceedings of the 8th ACM International Symposium on Mobile Ad hoc Networking and Computing, pp. 11–21. ACM Press, New York (2007)
19. Tolle, G., Culler, D.: Design of an Application-Cooperative Management System for Wireless Sensor Networks. In: Proceedings of the Second European Workshop on Wireless Sensor Networks (EWSN), pp. 121–132. IEEE Operations Center (2005)
20. Turau, V., Renner, C., Venzke, M.: The heathland experiment: Results and experiences. In: Proceedings of the REALWSN 2005 Workshop on Real-World Wireless Sensor Networks (June 2005)
21. Turon, M.: Mote-view: a sensor network monitoring and management tool. In: IEEE Workshop on Embedded Networked Sensors, pp.11–17 (2005)
22. Yuan, F., Song, W.-Z., Peterson, N., Peng, Y., Wang, L., Shirazi, B., LaHusen, R.: A lightweight sensor network management system design. In: IEEE International Conference on Pervasive Computing and Communications, pp. 288–293 (2008)
23. Zennaro, M., Ntareme, H., Bagula, A.: Experimental evaluation of temporal and energy characteristics of an outdoor sensor network. In: Mobility 2008: Proceedings of the International Conference on Mobile Technology, Applications, and Systems, pp. 1–5. ACM, New York (2008)